**TP N°3**

**Exercice 1**

Quelles sont les réponses de Prolog aux buts suivants :

?- [X,a|Y] = [c,a,a|K].

?- [X,a|Y] = [c,c,a|K].

?- [1|X] = [1,2,3,5].

?- [1,2|X] = [1,2].

?- [1,2|X] = [1,2,7,3,4].

?- [X] = [ ].

?- [X] = [1,2].

?- [X,Y|Z] = [1,2].

**Exercice 2**

Interroger l’interpréteur Prolog pour :

* Calculer la longueur d'une liste
* Afficher le dernier élément d'une liste
* Inverser une liste d'entiers
* Afficher le nième élément d'une liste
* Supprimer le dernier élément d'une liste
* Concaténer deux listes d'entiers

**Exercice 3**

Donner les buts permettant de :

* Trier une liste d'entiers de façon ascendante puis descendante.
* Afficher le dernier élément d’une liste en utilisant le prédicat append
* Supprimer une valeur donnée dans une liste

**Exercice 4 : (la récursivité)**

Définir un prédicat récursif pour chacun des traitements suivants :

* Calculer la longueur d'une liste
* Afficher le dernier élément d'une liste
* Inverser une liste d'entiers
* Afficher une suite numerique
* Afficher le nième élément d'une liste
* Supprimer une valeur dans une liste
* Supprimer le dernier élément d’une liste
* Concaténer deux listes
* Calculer le Nième terme de la suite de Fibonacci : $ u\_{0}= 1 , u\_{1} = 1 et u\_{n} = u\_{n-1} + u\_{n-2},  ∀n\in N, n > 1$

**Quelques prédicats prédéfinis sur les listes**

is\_list(+Terme). % Réussit si *Terme* est une liste.

is\_set(+Terme). % Réussit si *Terme* est un set, c'est-à-dire une liste ne comportant pas de doublons.

length(+Liste, +Int). % Unifie Int avec la longueur de la liste. Peut également servir à créer une liste de longueur Int ne contenant que des variables libres.

nth0(?Ind, ?List, ?Elem). % Réussit si l'élément d'indice Ind de la liste List s'unifie avec Elem. Les indices commencent à 0. sert généralement à récupérer l'élément d'indice Ind dans la liste List.

 Exemples : nth0(2, [3,1,8,6,9], E).

 nth0(2, [3,1,8,6,9], 8).

 nth0(I, [3,15,8], E).

nth1(?Ind, ?List, ?Elem*).% Pareil que nth0, mais avec les indices commençant à 1.*

last(?List, ?Elem). *% Unifie Elem avec le dernier élément de la liste List.*

numlist(+Min, +Max, -List). *% Unifie List avec la liste des entiers compris entre Min et Max.*

reverse(+List1, -List2). *% Inverse l'ordre des éléments de List1 et unifie le résultat dans List2.*

append(?List1, ?List2, ?List3*).% Réussit si List3 est la concaténation de List1 et List2. Il sert principalement pour ajouter le contenu de List2 à la suite de List1 et unifier le résultat dans List3.*

flatten(+List1, -List2*).% Aplanit List1 et unifie le résultat dans List2. List1 peut contenir de nombreuses listes imbriquées récursivement. flatten/2 extrait tous les éléments contenus dans List1 et stocke le résultat dans une liste « plane » (à une seule dimension).*

Exemple flatten([[1,[2],3], [[4,5],[6,7]]], Flat).

 Flat = [1, 2, 3, 4, 5, 6, 7]

permutation(?List1, ?List2). *% Permet de vérifier si List1 est une permutation de List2 ou inversement.*

member(?Elem, ?List).% *Réussit si*Elem*avec un élément de la liste*List*. Utilisé principalement pour énumérer les membres de la liste.*

msort(+List, -Triee). *% Trie la liste List et unifie le résultat dans Triee.*

sort(+List, -Triee). *%Se comporte comme msort/2, mais supprime les doublons.*

list\_to\_set(+List, -Set). *% Transforme une liste en set. En d'autres termes, supprime tous les doublons de List et unifie le résultat dans Set.*

sumlist(+List, -Sum). *% Unifie Sum avec la somme de tous les éléments de List.*

merge(+List1, +List2, -List3). *% List1 et List2 sont des listes triées. merge/3 fusionne List1 et List2 pour en faire en une liste triée et unifie le résultat dans List3. merge/3 ne supprime pas les doublons.*